Euryale

Kavitha Ranganathan krangana@cs.uchicago.edu
Catalin Dumitrescu cldumitr@cs.uchicago.edu
Jens-S. Vöckler voeckler@cs.uchicago.edu
Michael Wilde wilde@mcs.anl.gov

12/10/2003

Author	Date	Modification
Jens Vöckler	20031210	initial document
Jens Vöckler	20040126	Additions for log files
Jens Vöckler	20040406	Documenting new properties
Jens Vöckler	20040420	Adding a few paragraphs
Jens Vöckler	20040614	Adding some missing features

Contents

1 Overview

The Euryale system is a somewhat complex system to accomplish a complex task: Run jobs in the grid. While many such tools exist today, Euryale tries to take a late binding approach with a dash of fault tolerance through replanning, separately for each job.

Figure 1 shows the system overview as a Yourdon Flow chart with data- and control flows.

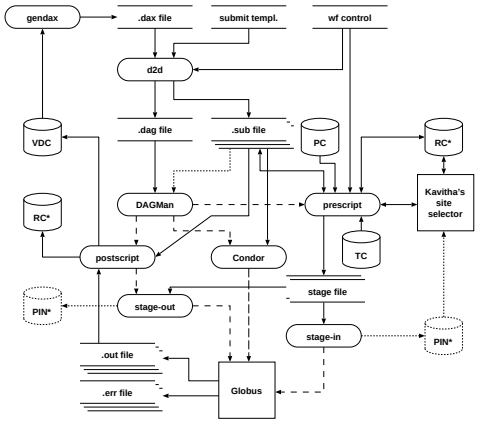

Table 1: Overview over planning workflow.

The key players are the DAX to DAG tool *d2d*, the job's *prescript* and the job's *postscript*. Each will be described in detail in sections of its own.

From the Virtual Data Catalog (VDC), a user may chose to materialize one or more data products, which includes their dependent computations. The resulting `.dax` file contains the workflow necessary to materialize the products, but at a pure logical level. Thus, the workflow will usually go back to those input files or generator jobs which don't have a predecessor known to the VDC. It is possible, though, to restrict the depth of the depth-first search tree.

It is also possible for the requested data products to form a disconnected graph – which means that more parallelism can be achieved.

2 From DAX to DAG

The `.dax` file is converted by the `d2d` tool into a Condor DAGMan `.dag` file, and a number of related Condor `.sub` submit files. The submit files are referenced by the `.dag` file.

The `d2d` converter bridges between the GVDS logical router (Chimera), and the Euryale logic. Thus, it is controlled by two property files: The GVDS properties, described elsewhere, and the workflow control properties, described in section 3.1 (page 5).

The submit files' skeleton is based on substitutions from a submit file template `.sft`. The template contains a number of placeholders for different purposes. Through the templating mechanism, the workflow can be easily adapted to different site requirements.

The template contains two kinds of variables, `@@variable@@` and `!!variable!!` holes. The first kind of variables are substituted by the `d2d` tool. Some variables substitute to their final value, some to an empty string, and some to `!!variable!!` for a second round of deferred substitutions at the late planning point. Table 2 shows the recognized identifiers and substitutions during the early phase.

variable	substitution
ARGS	taken from the job's arguments, may be empty.
CONFIG	translates into the name of the k.2 configuration file.
DAXLABEL	value from XML `adag@label` attribute. It is an arbitrary label chosen by the user at DAX creation time.
DAXMTIME	is the last modification time of the `.dax` file.
DV	value from the combined XML `job@dv-*` attributes. This value names the VDL derivation.
JOBID	value from XML `adag/job@id` attribute. Each ID is unique to the workflow, but will repeat between workflows.
LEVEL	value from XML `adag/job@level` attribute. The level is the distance from the requested data product.
MAXPEND	value from `wf.max.pending` property: The maximum number of seconds a job is willing to spend in Condor's local *idle* state, which is equivalent to remote pending, until the job is killed for replanning. This value defaults to 2 hours. A minimum of 10 minutes is enforced.
STDIN	translates into a later substituted LFN, possibly empty, from the XML `adag/job/stdin` element.
STDERR	translates into a later substituted LFN, possibly empty, from the XML `adag/job/stderr` element.
STDOUT	translates into a later substituted LFN, possibly empty, from the XML `adag/job/stdout` element.
SUBMIT	becomes the name of the submit file itself.
SUBBASE	becomes the submit filename minus the `.sub` suffix.
TEMPLATE	is the submit file template `.sft` filename.
TR	value from the combined XML `job` attributes `namespace`, `name`, and `version`. This value names the VDL transformation.
VERSION	for starters 1.2 will do.

Table 2: Early @@variable@@ substitutions.

The `d2d` transformer uses a variety of commandline arguments to control the output. Except for the location of the `.sft` file and the `.dax` input file, these arguments are optional.

The `--dir` option takes as argument the directory into which the new files are to be generated. A user should use a separate directory for each workflow. While the converter warns about existing files, it will

still overwrite them. By default, all output is generated in the current working directory, which is usually *not* the user's intention.

The `--prescript` option allows to override the location of the file to be inserted as DAGMan prescript. By default, the position is determined from the workflow configuration file. Similarily works the `--postscript` option.

The `--wfrc` option takes as argument the location of the workflow configuration file. This file is a Java property-style file. It allows to configure multiple aspects of the conversion, as well as aspects during runtime. By default, a file `.wfrc` is read from the user's |$HOME— directory.

The `--template` option is mandatory, and points to the location of the submit file template. This option allows for easy exchange of compute job characteristics for different pool sets, e.g. Grid3 versus LCG1.

Any other argument not documented here is experimental, and may not work.

The final argument is the location of the `.dax` file to convert. This file contains the workflow.

Please note that no optimizations are attempted. Since the `.dax` file contains usually all necessary workflows to generate a data product, so will all the computations present in the `.dax` file be executed.

3 Configuration Files

3.1 Workflow Properties (WF)

All workflow configuration properties start with the constant prefix `wf<dot>`. This prefix will be omitted in the discussion of table 3.

key	value	meaning
site.selector	*path*	Absolute path to the site selector.
site.temp.suffix	*string*	Filename suffix of the temporary file that will generated for the site selector. Defaults to `.lof`.
site.temp.dir	*dir*	Location of the directory to generate temporary files in. Defaults to the system's default `/tmp` on most platforms.
remote.job.queues	*kv list*	Comma-separated list of key=value pairs. Each keys represents a site handle, and each value the name of the queue to use for that site.
remote.job.projects	*kv list*	Comma-separated list of key=value pairs. Each keys represents a site handle, and each value the name of the project to use for that site.
popularity.manager	*path*	Absolute path to the popularity manager application.
site.temp.unlink		
keep.rewrite	*boolean*	Keep backups of raw submit files after successful job runs.
script.pre	*path*	Absolute path to the prescript.
script.post	*path*	Absolute path to the postscript.

continued on next page

continued from last page

key	value	meaning
replica.pin		Future: File pinning hook.
pool.style	old	Format of the pool configuration. The file is compatible with the Pegasus old textual pool.config file. The only supported style is ”old”.
pool.file	*path*	Absolute path to the pool.config file.
tc.style	old	Format of the transformation catalog. The file is compatible with the Pegasus old textual pool.config file. The only supported style is ”old”.
tc.file	*path*	Absolute path to the tc.data file.
rc.style	Dbm LRC	Style of the replica catalog interface.
rc.file	*path*	In BSD-DB mode, this is the absolute path to the database containing the replica catalog.
rc.lrc	*URI*	The contact for your local LRC.
rc.java.home	*dir*	The directory where Java is installed. Defaults to $JAVA_HOME.
rc.globus.location	*dir*	The directory where Globus is installed. Defaults to $GLOBUS_LOCATION.
rc.grc	*path*	Absolute path to Globus’ `globus-rls-cli`.
rc.r_c	*path*	Absolute path to the GVDS `rls-client`.
job.retries	*positive*	Number of retries for failed compute jobs. This also applies to failed post-jobs.
site.temp.unlink	*boolean*	Remove temporary files after done. Defaults to `true`.
site.temp.suffix	*suffix*	The suffix to use when creating list of files in temporary files. Defaults to suffix `.lof`.
site.temp.dir	*dir*	The directory where to generate temporary files. Defaults to the value of Java property `io.tmp`. Try `/dev/shm` on modern Linuxes.
bad.timeout	*integer*	The timeout until a bad site is retried. This is an experimental feature yet, which has no effect on production levels.
cache.manager	*path*	The name of the cache manager application. This is a data scheduling addition. Defaults to a noop application.
data.scheduler	*path*	The name of the data scheduling application. This is a data scheduling addition. Defaults to a noop.
popularity.table	*path*	The location of a file which maintains the popularity of files. This is a data scheduling addition.

Table 3: Properties to configuration a workflow.

The workflow control file contains the configurations and user-selectable options for all pieces of the Euryale framework. Using the simple properties permits various call-outs, and user-provided implementation

of Euryale's abstract interfaces.

3.2 Pool Configuration (PC)

[describe interface]

[currently only old textual style, rDBMS possible]

3.3 Transformation Catalog (TC)

[describe interface]

[currently only old single-line textual style, XML/rDBMS possible]

3.4 Replica Catalog (RC)

[describe interface]

[currenly either local BSD-DB file, or RLS-LRC service]

4 Execution

The generated `.dag` with its submit files can be executed by DAGMan. DAGMan will in turn invoke the prescript, which calls out to Kavitha's site selector (KSS[1]). The KSS logic selects one from the number of available sites.

The original site set is determined by the sites present in the PC. The set of sites is further limited by those sites in the TC, which have the computation available to them. The site selector choses a site from a given list depending on a variety of strategies. The simplest strategy is pure random, which works well for trying out things, and an always-the-same, which is good for testing a specific site. Fancier algorithms, which take the presence of data at the destination site into account, are presented separately by Kavitha and Catalin.

For the selected site, the submit file is finalized by substituting the `!!var!!` variables. For the substitution, all profiles are combined, with the VDL profiles at lowest priority, the PC profiles next, the TC profiles above that, and potential user profiles to override any lower priorities. Detailled overwrite, merge, or replace handling is not implemented - a higher priority overwrite a lower priority value.

Input files are transferred always using submit-host-initiated 3rd party transfers. If the transfers fail for any reason, replanning takes place, marking the failed site as "bad" for the job. Among the fatal workflow error condition is running out of runnable sites.

If the prescript ran successfully, DAGMan will start the compute job via Condor-G. Once the compute job finished, DAGMan will run the post script, which parses the kickstart record, and transfers output files.

[1] There are several ones available. The interface is file-driven and simple, see 4.2.1

Currently, it will leave output files just where they are. There is no central collection, as Kavitha research an asynchroneous data scheduler.

If the post script determines that a transfer failed, or the job failed, the postscript will fail, and DAGMan will up to the number of RETRIES attempt to run the job again, starting with the prescript. The postscript also tagged the failed site as ”bad” to keep the KSS from chosing it again. There is no central knowledge of bad sites (yet).

4.1 Debugging and Logfiles

Pre- and postscript protocol verbosely what they are doing in a per job `.dbg` file. Each jobs also logs certain information of global interest into a common, shared `euryale.log` file.

4.1.1 The Common Logfile

Currently, the common logfile name is hard-coded as `euryale.log`. The common log is required for things like compute time, number of transfers, etc. One such file is associated with each workflow. For reasons of performance, the information in this file is few and restricted. Future version may use the workflow label to determine a more unique filename to avoid common log filename clashes.

#	meaning
1	log event tag to aide later parsing: tag: meaning prs: prescript started prf: prescript finished pos: postscript started prf: postscript finished si1: stage-in report sum: remote exitcode
2	ISO 8601 timestamp, local zone offset omitted.
3	process ID of the logging script in brackets.
4	unique job identifier.
5	PRE for prescript, POST for postscript entries.
6	free format log entry.

Table 4: Workflow-common log file.

The central logfile contains its entries in 6 or more columns, one entry per log incidence. Table 4 shows the layout of the common log file.

4.1.2 The Job’s Logfile

Each job protocols extensive debug information into a job-specific log file. The name of the job-specific log file is derived from the name of the submit file. The extension `.sub` is replaced with `.dbg` for the

job logfile.

#	meaning
1	ISO 8601 timestamp, local zone offset omitted.
2	process ID of the logging script in brackets.
3	PRE for prescript, POST for postscript entries.
4	free format log entry.

Table 5: Job-specific log file.

The layout of the job logfile is similar to the layout of the common logfile. Its particular layout is shown in table 5. There are no incidence tags, as the number of log messages is proliferate. Since the log file is specific for a single compute job, it will not repeat the job identifier, either.

If problems are experienced, the job logfile usually contains clues as to why some things happen. For instance, a changing pid is an indication of replanning. Often, the reason for replanning can be gleaned from the information in the debug file, too.

4.2 The Prescript

variable	substitution
WORKDIR	value to be substituted with the working directory to use on the remote site's worker node. This is taken from the 5th column of the pool configuration
SITE	pool handle for the chosen site, see pool configuration 3.2. This is the 1st column from the pool configuration.
KICKSTART	the location of the remote application launch utility. This is the 6th column of the pool configuration.
TRANSFORMATION	will be substituted with the logical transformation name.
APPLICATION	will be substituted with the remote application's location.
GLOBUSSCHEDULER	is the jobmanager contact string for the remote site. This is the 3rd column from the pool configuration.
STDIO	In case a job tracks any of its stdio, this will convert to the appropriate kickstart invocation arguments.
GLOBUSRSL	will extract from all profiles the the `globus` namespace, and convert values into RSL strings.
CONDOR_GLOBUSRSL	same as GLOBUSRSL, except that the result is prefixed with `globusrsl =` , and empty RSL will not generate any line.
CONDOR_ADDON	will extract values from the `condor` profile namespace. This allows for job-specific Condor additions.
ENVIRONMENT	will extract values from the `env` profile namespace to generate a common environment setting.
LFN:xxx	This special key will substitute the logical filename `xxxx` with its storage filename (SFN).

continued on next page

continued from last page

variable	substitution
ENV:xxx	This special key will substitute the environement value for key `xxxx` in the submit hosts environment.

Table 6: Late !!variable!! substitutions.

4.2.1 Kavitha's Site Selector

The site to run a compute job is selected dynamically at the point of time a compute job becomes runnable. The dynamic site selector is coded via a call-out to an external program. To talk to the site selector, a temporary file is generated, which contains two sections, for example:

```
[filenames]
krangana.f.b

[sites]
UM_ATLAS     gsiftp://atgrid.grid.umich.edu/
IU_ATLAS_Tier2 gsiftp://atlas.iu.edu/
UBuffalo_CCR gsiftp://acdc.ccr.buffalo.edu/
```

The first section lists the input filenames for the given computation. The second section lists pool handles, followed by the gridftp server base-URI for that particular pool. Note that only the gridftp hostname, and potentially a port, will be listed.

The prescript creates the above data in a temporary filename. It calls out to the site selector application specified in the WF property `wf.site.selector`, passing the name of the temporary file as its argument. The site selector in turn prints a string `SOLUTION:xxx` on its *stdout*, where *xxx* is the site handle that was chosen.

The chosen site must be a site handle from the second section. During replanning of jobs, bad sites are cached for each compute job, and omitted from the list of sites passed to the site selector. If no valid compute sites can be found, the prescript exits with an error.

The beauty of the external site selector is that it can be coded in any programming language that is natively runnable in the host environment, usually any scripting or compiled language[2]

A demo site selector, which randomly choses from the given site set, is included in the distribution. More elaborate schemes allow the integration of file popularity and monitoring feedback.

The information contained in the temporary communication file between Euryale logic and external site selector may be subject to extension in the future.

[2] Java classes work with Linux, if the `binfmt` kernel trappings are appropriately configured.

4.3 The Postscript

The post script logic parser the information returned by the external kickstart file for job success. However, it will not, at this point, fill information into the provenance tracking subcatalog of the VDC.

If the postscript determines that an execution failed, it will fail itself, and thus have DAGMan restart the work with the prescript. Such retry will have marked the failed site as locally bad, excluding it from replanning.

The postscript may move files from the current working directory to the site's storage element or directy, using a submit-host initiated 3rd party copy. However, result files will remain at the site they were produced. Euryale does not (yet) move results that are dear to the user, and tagged as such, to any output storage element, or list thereof.

4.3.1 Kavitha's Popularity Manager (KPM)

Each compute job that ran to a successful completion will update the popularity manager for input files. To talk to the dynamically popularity manager, a temporary file is generated, which contains just the list of logical input filenames. Note that this file may be empty.

The popularity manager is invoked with the site and the file of files as commandline arguments. It is not expected to return anything. The default popularity manager is a no-operation. More elaborate popularity managers can update the popularity of a file, and thus asynchroneously push hot-spot files to likely candidates. Or they can just keep statistics.